Pro/file Updates

The Newsletter For ZX Pro/file Users

Vol. 2, No. 4 October 1985

TURNING POINT

This issue of Pro/file Updates closes out the newsletter's second year. I want to thank all of you for your generous and enthusiastic support. It was your suggestions, contributions, and even complaints that made the first two years of Updates a smashing success. And I think it is safe to say that the ZX Pro/File program is now much improved over its former self with all the enhancements the newsletter has offered.

The many articles and improvements have, however, created a problem. I am running out of things to say (incredible, but true). After I put together the last issue with the machine code alphabetizing routine it suddenly dawned on me, that article would be a tough act to follow. What was I to do for an encore? Would there even be an encore? Would there be a future for Pro/File Updates?

The thought of ending this newsletter saddened me because I enjoy putting it out. But I just couldn't see any big new improvements coming up, and without them there's not much point in having a newsletter. As I agonized over this dilemma, one reader--and I wish I could remember his name--offered the perfect solution. He said, why not broaden the scope of Updates to include all of the products I sell (Hmm, I thought, fresh topics to write about...). Keep it quarterly, increase the price to $16.95 per year (sounds better all the time...), and increase the number of pages to 16 per issue (...with only one minor draw-back).

I mentioned this idea to several of you who called. Everybody seemed to think it was a good idea. Therefore, the Pro/file Updates as we know it shall end with this issue. In its place will be "COMPUTER UPDATES"; a broader name for a broader scope. COMPUTER UPDATES will continue as a quarterly publication. Subject matter will include improvements, extensions, ideas, tips, applications, program listings, and descriptions of the ZX/TS products I sell. I am not sure yet how many pages the new UPDATES will be, but it will definitely be larger that the present 8 page format. More pages, unfortunately, means more printing and postage costs, plus more time researching writing and typing. To cover this, the price for COMPUTER UPDATES must go up. But not a lot. The new subscription price shall be $12.95. This is just $3 more than last year. $3 is equal to 6 cups of coffee, 1 ticket to a lousy movie, or 1-1/2 boxes of Cheerios.

Will you get your money's worth? Well, I'll let you decide. Inside this issue I'm including a sample of articles such as you would find in the new COMPUTER UPDATES. Read them. If you like them, subscribe. I think you'll be very pleased.

HOW TO CONNECT YOUR TS/ZX TO A MONITOR--EASILY

by Tom Laffin

Anyone who has stared at those funny little moving marks or lines of distortion on TV displays, anyone who would like to enjoy a crystal clear image without the annoyance of RF interference and having to carefully adjust the TV set's fine tuning, or arrange cables just so, anyone who thinks a quality monitor would put the little devil into the professional class (and put an end to eye shock treatments), has GOT to try this.

Purchase a quality nine inch video monitor. Make up a shielded cable using RG-174 or other high quality micromini coaxial cable. Open the case of your TS/ZX and observe three wires bent in a "U" shape coming out one side of the RF modulator (the tin can that your TV cable plugs into). The wire nearest the outside edge can be tapped into, with the center conductor of your shielded cable. Connect the braid of your cable to the circuit board ground. This is a place where the tin can has tabs coming through the printed circuit and is soldered underneath.

Feed your cable out through the hole next to where you would plug in the TV cable and forget using that TV jack any more. (If you attempt to use the TV cable, you might find the display a little lighter, due to circuit loading caused by the capacitance of the direct video connection.)

If your monitor has a switch for 75 ohm termination, or bridging, try it either way for the best picture. It doesn't matter to the TS/ZX if the direct connection is terminated or not.

PRO/FILE PLUS CASSETTE CLARIFICATION

Last issue I stated that cassettes of "ZX" which contained many enhancements published in Updates was available. Apparently it was not clear to many that the biggest enhancement of all, the machine code alphabetizing routine featured in the last issue, was included on this tape. Price is $16.95 + $1.50 postage. Send to me, Tom Woods, P.O. Box 64, Jefferson, NH 03583.

PRO/FILE POKES FOR THE BYTE-BACK CENTRONICS PRINTER I/F

by Gloria Gedonious

Here is the Parallel I/F modification for the new Byte-Back interface. Load Pro/File and alter the printer code with:

```
POKE 16674,6
POKE 16675,0
POKE 16676,205
POKE 16677,105
POKE 16678,32
POKE 16679,42
POKE 16680,14
POKE 16681,64
POKE 16682,195
POKE 16683,138
POKE 16684,32
```

Change these basic lines:

```
3010 POKE 16675,C2
3020 RAND USR 16674
3036 GOSUB 3500
```

Add these lines:

```
3500 CLS
3510 PRINT
3520 RAND USR (your print address)
3521 RETURN
```

BASIC BYTES FREE

by David Youngquist

When modifying ZX Pro/File it is nice to see how many BASIC bytes are free. The main menu can show this number with a short machine code routine of 13 bytes. Space is needed in the 8-16K region or in the D$ (see Updates vol.1, #2, pg.5). Poke these numbers in succesive addresses:

33,0,0,57,237,91,28,64,237,82,68,77,201

Then change line 18 to:

```
 18 PRINT "XZ PRO/FILE BASIC
 BYTES ";USR (address of code);
,,,"ENTER A SEARCH COMMAND"---
-and so forth.
```

Z80-B, BENT EPROMS, ROM-PAK NOTES from W. Komlosy

Please note that the ROMPAK edition of ZX Pro/File has many program lines (all the DIM lines plus 9960/9970) which can be deleted after the program is transferred from eprom to memory and before the program is saved to tape. This saved memory can be used to enlarge D$ or add modifications.

By the way, that tip to replace the CPU with a Z80-B sure works fine here. I've got 16 2716 EPROMS and 12 6116 SRAMS (on 4 reworked "Hunter Boards), all plugged into the printer interface that is plugged into the rear of my computer without any buffering at all. Quite a load on the Z80-B when you consider that on the TS1000 PC board I have piggy-backed in the 2K RAM memory decoding and Oliger circuit for M.C. in high memory. That means 72K of memory and the improved 8K ROM all are loading down the CPU address lines constantly.

I never did get the improved display options to work right even with the fix in Vol.2,#1. Errors in the Updates (which should be edited out before printing-Ha Ha) should at least be corrected in next issues.

---Dear W. Komlosy, If I had that much stuff plugged into MY rear I would want some kind of buffering too. As regards to errors and the Display Option improvements, keep trying. You'll get it one of these days. Many have successfully made the mod. Typographical errors in computer programs are an editor's nightmare. I try my best and will continue to do so. After each issue goes to the printer, boy do I sweat it out until I hear from a reader that they got a modification working. A final note about the ROM PAK version of "ZX": ROMPAK is now officially out of business. This is probably a good thing because the service they gave would lead one to believe that they were out of business long before they really stopped taking people's money. The good news is that Sunset Electronics is taking over the ROMPAK line. With a fine company like Sunset, you can be sure you'll get what you pay for and there'll be help when you need it. Sunset is located at 2254 Caraval St., San Francisco, CA 94116. They have a monsterous catalog. Write for one.

AUTOLOG, an extensive modification to ZX Pro/File is available from Communications Systems Center, Bridge St. Box 133, Hillsboro, NH 03244. AUTOLOG allows for automatic entry of single line files, printing as files are added, and resetting D$ when memory is full. AUTOLOG is used as a police and fire department's radio and phone log, and at an alarm center to record periodic clocks. Write for info.

INTRODUCING THE EXPERIMENTOR'S UNIVERSAL INPUT/OUTPUT PORT

Some of you may remember that a LONG time ago, I said I wanted to offer a simple parallel port and use it as the basis of a series of programs designed to show computer hobbyists the what's, why's and wherefore's of "Programming for Ports". My plan was (and still is) to provide many interesting applications centering around this I/O port, and then through the documentation, explain how the program works so that you can glean information on programming techniques you can use in your own programs. After many months of design work, by both myself and John Oliger whom I contracted with to manufacture the port boards, as well as considerable time writing the first few applications, I am happy to say the product is ready to be launched.

PORT FUNDAMENTALS

If you are wondering what in the world a "port" is and why you would ever want one, let me introduce you to a few fundamentals. A "port" is a device which plugs into your

Pro/file Updates is published 4 times a year, in January, April, July, and October.

Subsciption rate is $9.95 annually.

Edited and Published by:

Thomas B. Woods
P.O. Box 64, Jefferson, NH 03583
(603) 586-7734

computer which makes it possible to exchange data with other devices. Your computer's mic and ear jacks are connected to a cassette port built right in to the computer. When you LOAD this port takes data from your tape recorder and puts it into the computer's memory. When you SAVE a program, you do the reverse. Data from memory goes out to the recorder. The picture generated by the computer goes out to the TV through a port. Data you type when you press on the keyboard goes into memory through a port.

Ports are like nerves. Some connect sensory organs with the brain. Others serve as the conduit through which the brain sends impulses to muscles telling them to act.

Through ports you can connect motors, temperature probes, electric eyes, vibration sensors, relays, sound instruments, chemical sniffers, lights, and other "real world" devices up to your computer. The data transmitted to the computer through the input ports can be analyzed and then signals can be sent out to turn on other devices as a response.

Imagine a completely computer controlled solar heating system, a motorized robot, a fully automatic weather station, an alarm system, economical and accurate lab testing equipment. Or how about computerized guidance systems for telescopes, cameras, or video recorders. These are some of the things you can do with ports.

Unfortunately, there is a giant gap between imagining these systems and actually implementing them. Programming for ports is a science unto itself. It is a fairly simple matter to get signals into or out of the computer, but interpreting those signals and knowing if the interpretation is accurate is another matter entirely. The computer's eye view of the world presents quite a different reality from everyday affairs. When you program a computer to sense and act on outside events, you must look at the world through the eyes of the computer. To me, its rather like looking through a microscope or some other device which throws a completely different light on your perception of things.

ABOUT THE EXPERIMENTOR'S I/O PORT

The Experimentor's Universal Input/Output Port was designed to be useful in many different settings. It will plug into the back of the TS1000/1500, the TS2068, or the Spectrum without any modifications to either the port board or the computer. The board has 8 input lines and 8 latched output lines. This means that a binary series of highs and lows sent to the outputs will remain there until you change them by sending a different series of highs and lows. In addition there are two strobe lines provided. These output lines are active when the computer is inputting or outputting data to the port. They can be used for various forms of "handshaking".

To make this port board easy for the experimentor to configure and reconfigure many times for different applications, connections to the inputs and outputs are provided by way of two 10 position female sockets. One socket is for the 8 input lines, a ground, and a strobe line. The other socket is the same for the output lines. These sockets allow for temporary solderless breadboard type hook-ups. You can also make your connections with the common 1/10th inch center male headers, or by using crimp-on male connector pins (a supply of these is included with the unit).

Ports are like memory locations in that each port must have a unique address. This board is given the decimal address of 223 (or DF in hex). It is quite easy to change this address in the unlikely event you have a peripheral which already uses this address.

THE BASIC I/O TECHNIQUE

The actual technique of reading values from or writing values to this port varies with both the program and the computer you use. With the Spectrum and TS2068 you can use the Basic IN and OUT commands:

```
LET X=IN 223
```

will read the 8 input lines and put the value it finds into the variable X.

OUT 223,X

nds the variable X out to port number 223. These commands are very similar to the PEEK and POKE commands except that instead of reading or writing to a memory location, you read or write to a port.

On the TS1000/1500 there is no Basic command which will access ports. Instead, you must use machine code. (I should mention, however, that this will not be true for long. I expect to have a new program soon which extends the Sinclair Basic with 22 new commands. Among them are IN and OUT which work in the same way as those on the TS2068). Perhaps the simplest machine code routine you'll ever get from me is the one below which lets the TS1000 access this port:

```
LD A,00
OUT (DF),A
RET
IN A,(DF)
LD C,A
LD B,00
RET
```

To enter this into your computer, create a REM line consisting of at least 11 spaces after the REM token. Then make the following pokes:

```
POKE 16514,62
POKE 16515,0
POKE 16516,211
POKE 16517,223
POKE 16518,201
POKE 16519,219
POKE 16520,223
POKE 16521,79
POKE 16522,6
POKE 16523,0
POKE 16524,201
```

Write to port 223 by poking 16515 with the value you wish to output. Then, RAND USR 16514. If you connect the port up to the computer and execute the commands below:

```
POKE 16515,255
RAND USR 16514
```

u could use a voltmeter to measure the voltage at all 8 output connections and find that each one measures 5 volts. Then if you POKE 16515,1 and RAND USR 16514 the voltage at bit 0 would still be high, but all the others would be low.

Reading data from the input port is accomplished by entering the command: LET X=USR 16519. When the machine code executes, it reads the instantaneous state of the 8 input bits and stores the decimal value in the variable X. If all the inputs are connected to ground, X will equal 0. If they are all tied to 5 volts, X will equal 255.

PORT PROJECT #1
DATA COMMUNICATIONS BETWEEN TWO COMPUTERS

This first project gives you the basis of a very useful application involving I/O ports--communicating between two computers. Some benefits this ability gives are:

*Sharing data between two separate computers
*Moving data from the TS1000 into the TS2068
*Increasing RAM capacity
*Send data to be lprinted or saved to a disk drive to the second computer thereby allowing the first computer to operate on something else.
*Use two machine code programs which occupy the same area of memory by storing one in the first computer while the other program is being executed by the second computer.

You end up with two independent computers able to run separate programs and able to share common data. In the case of increasing ram capacity, the second computer can serve as a huge storage medium something like a RAM-DISK. Its almost like being able to plug a second 64K ram pack into your computer.

This project connects the TS1000 with the TS2068. It is quite possible to hook two 1000's or two 2068's together, but you must relocate the machine code to a different area of memory. Before you can start this project, you must already have a few items. First, you need two port boards (one for each computer). While this system uses the

Experimentor Port Boards mentioned here, there is nothing to prevent you from using other commercially available ports, or boards of your own design. All you need is 8 bits of latched output and 8 bits of input. You also need a 64K ram pack for the TS1000 (one that gives you memory in the 8-16K block). I used a Byte-Back UM64 memory. Because the UM64 lets you switch this block in or out in 2K increments, you can put the machine code in the 8-10K area and still have the 10-12K block free for the Memotech printer interface. The Memotech 64K ram pack won't let you do this. If you don't use the Memotech I/F this is not a problem.

To visualize how this system works, I use the "master/slave" concept. One computer (the 2068) serves as the master. The other (TS1000) is the slave. The job of the master computer is to keep track of what is stored in the slave's memory, and to issue commands to the slave. The job of the slave is to patiently sit and wait for commands from the master. Then, to perform the command specified.

The machine code shown here provides a rudimentary set of routines which accomplish this. As listed, the master has three commands it can send: SEND memory to the slave, RECEIVE memory from the slave, and cause the slave to QUIT to Basic. More commands will be developed later as needs become apparent. Up to 128 different commands could be implemented.

Two separate programs are required. One for the master computer, one for the slave. Type listing 1. into the computer you designate as the slave and RUN it. You will be asked to enter the values shown in listing 2. As you do this, the computer puts the value you input into an A$ array. The remainder of the program moves this machine code down to address 8192, and lets you make a tape back up of the code in the REM line. You would load this tape any time you want to use the system.

Listing 1. TS1000 Slave Basic Input Routine

```
  1 FAST
  5 LET N=0
 10 DIM A$(518)
 20 FOR X=1 TO 209
 25 SCROLL
 30 PRINT X;"=?";
 40 INPUT Y
 50 PRINT Y
 55 LET A$(X)=CHR$ Y
 57 LET N=N+Y
 60 NEXT X
 70 LET A$(255 TO 261)=CHR$ 247
+CHR$ 32+CHR$ 129+CHR$ 32+CHR$ 1
58+CHR$ 32+CHR$ 189
 80 FOR X=262 TO 513 STEP 2
 90 LET A$(X TO X+1)=CHR$ 32+CH
R$ 70
100 NEXT X
110 CLS
115 PRINT "SUM OF CODES INPUT I
S:";N;TAB 0;"SHOULD BE:23017"
117 IF N<>23017 THEN PRINT "***
120 PRINT "CODE INPUT IS COMPLE
TE.          PRESS ENTER TO MAKE
TAPE BACK-UP"
122 INPUT X$
126 SAVE "SLAVE"
128 SLOW
130 PRINT "NOW MOVING CODE DOWN
TO 8192"
140 LET Z=8192
150 FOR X=1 TO 515
160 POKE Z,CODE A$(X)
170 LET Z=Z+1
180 NEXT X
190 PRINT
200 PRINT "TRANSFER IS COMPLETE
"
210 PRINT "PRESS ENTER TO START
THE SLAVE  SYSTEM"
220 INPUT X$
230 FAST
240 LET X=USR 8262
```

Listing 2. TS1000 Slave Machine Code Poker Table

```
1=197   22=32   43=192  64=119  85=241  106=94  127=35  148=32  169=205 190=235
2=      23=25   44=24   65=120  86=33   107=35  128=70  149=16  170=7   191=78
3=32    24=126  45=251  66=147  87=247  108=86  129=201 150=242 171=32  192=35
4=16    25=237  46=205  67=19   88=32   109=235 130=205 151=13  172=35  193=70
5=254   26=121  47=38   68=32   89=34   110=209 131=115 152=40  173=34  194=195
6=193   27=123  48=32   69=239  90=254  111=19  132=32  153=173 174=254 195=34
7=201   28=184  49=17   70=201  91=32   112=205 133=175 154=205 175=32  196=32
8=22    29=32   50=0    71=17   92=229  113=34  134=184 155=45  176=193 197=205
9=0     30=243  51=0    72=0    93=205  114=32  135=40  156=32  177=16  198=34
10=14   31=237  52=71   73=0    94=51   115=233 136=14  157=24  178=243 199=32
11=223  32=120  53=237  74=205  95=32   116=98  137=197 158=38  179=121 200=195
12=71   33=32   54=89   75=38   96=58   117=107 138=205 159=205 180=254 201=70
13=237  34=252  55=205  76=32   97=247  118=94  139=45  160=115 181=0   202=32
14=121  35=175  56=202  77=254  98=32   119=35  140=32  161=32  182=40  203=205
15=205  36=211  57=32   78=4    99=33   120=86  141=205 162=175 183=143 204=0
16=0    37=223  58=42   79=40   100=0   121=237 142=34  163=184 184=61  205=32
17=32   38=201  59=254  80=5    101=33  122=83  143=32  164=40  185=205 206=205
18=237  39=14   60=32   81=175  102=133 123=254 144=193 165=13  186=7   207=0
19=88   40=223  61=25   82=237  103=111 124=32  145=35  166=62  187=32  208=32
20=42   41=237  62=237  83=121  104=203 125=35  146=34  167=255 188=24  209=201
21=254  42=120  63=120  84=24   105=37  126=78  147=254 168=197 189=137
```

After everything is entered and saved, you can hook the computers together. All you have to do is plug the port boards into their respective computers (with the power turned OFF, of course) and make up a cable which connects each of the 8 input and output bits on one board to their opposite partners on the other. Inputs go to outputs, outputs go to inputs. Be sure that the data lines match properly. In other words, Output bit 0 should go to input bit 0. Bit 1 goes to bit 1, 2 goes to 2, etc. It is easy to get them mixed up. Radio Shack now sells spools of 25 conductor ribbon cable which is ideal for making these cables (cat.# output bits of one computer to the 8 input bits of the other.

When the connections are completed, load the slave tape into the slave computer. It will run automatically, placing the operating system down into low memory. Once the transfer is finished, FAST mode is selected and the computer goes into the slave state. Because the computer is in FAST, there is no display when this happens. The keyboard is also dead. At first, this can be a little disconcerting because it appears as though the computer has crashed. If you entered the machine code correctly however, do not be alarmed. The slave is just patiently awaiting instructions from the master. The keyboard and TV display are no longer needed. In fact, you don't even need to hook up the TV if you don't want to.

Now you're ready to load the master computer. Once it is in, your operating system is ready and waiting for you to use. You can integrate this system with other programs such as Pro/File 2068 or your own. The only thing you can't do is try to load machine code on top of the master operating system. In the case of Pro/File 2068, there is no conflict. Other programs may be different.

```
  1 REM MASTER INPUT PROGRAM
  5 CLEAR 64769
 20 FOR X=64770 TO 65029
 30 PRINT X;"=";
 40 INPUT "ADDRESS ";(X);"?",Y
 50 PRINT Y
 60 POKE X,Y
 70 LET N=N+Y
 80 NEXT X
 85 LET N=0: FOR X=64770 TO 650
29: LET N=N+PEEK X: NEXT X
 90 CLS : PRINT "CODE INPUT IS
COMPLETE": PRINT "SUM OF INPUT C
ODES IS:";N: PRINT "SHOULD BE:";
32590
 100 IF N<>32590 THEN PRINT FLAS
H 1;"WARNING"
 110 INPUT "PRESS ENTER TO SAVE
CODE";X$: SAVE "MASTER"CODE 6477
0,260
```

Listing 3. TS2068 Master Basic Input Routine

Listing 4. TS2068 Master Machine Code Poker Table

```
64770=197   64857=205   64944=0     64814=24    64901=126   64988=35
64771=6     64858=77    64945=235   64815=251   64902=254   64989=34
64772=32    64859=253   64946=237   64816=205   64903=9     64990=254
64773=16    64860=175   64947=176   64817=41    64904=32    64991=253
64774=254   64861=184   64948=27    64818=253   64905=248   64992=205
64775=193   64862=40    64949=27    64819=17    64906=35    64993=2
64776=201   64863=13    64950=35    64820=0     64907=126   64994=253
64777=243   64864=62    64951=35    64821=0     64908=254   64995=16
64778=22    64865=255   64952=235   64822=71    64909=0     64996=242
64779=0     64866=197   64953=14    64823=237   64910=32    64997=13
64780=14    64867=205   64954=2     64824=89    64911=242   64998=196
64781=223   64868=9     64955=237   64825=205   64912=35    64999=48
64782=71    64869=253   64956=176   64826=234   64913=126   65000=253
64783=237   64870=35    64957=235   64827=253   64914=229   65001=201
64784=121   64871=34    64958=17    64828=42    64915=33    65002=205
64785=205   64872=254   64959=4     64829=254   64916=0     65003=2
64786=2     64873=253   64960=0     64830=253   64917=254   65004=253
64787=253   64874=193   64961=237   64831=25    64918=133   65005=205
64788=237   64875=16    64962=82    64832=237   64919=111   65006=2
64789=88    64876=243   64963=123   64833=120   64920=203   65007=253
64790=42    64877=121   64964=229   64834=119   64921=37    65008=201
64791=254   64878=254   64965=205   64835=120   64922=94    65009=0
64792=253   64879=0     64966=9     64836=147   64923=35    65010=0
64793=25    64880=200   64967=253   64837=19    64924=86    65011=0
64794=126   64881=61    64968=225   64838=32    64925=235   65012=0
64795=237   64882=24    64969=201   64839=239   64926=209   65013=0
64796=121   64883=149   64970=205   64840=24    64927=19    65014=0
64797=123   64884=167   64971=166   64841=219   64928=233   65015=0
64798=184   64885=42    64972=253   64842=0     64929=205   65016=0
64799=32    64886=89    64973=0     64843=0     64930=166   65017=0
64800=243   64887=92    64974=0     64844=0     64931=253   65018=0
64801=237   64888=1     64975=0     64845=94    64932=24    65019=0
64802=120   64889=13    64976=205   64846=35    64933=179   65020=0
64803=32    64890=0     64977=77    64847=86    64934=33    65021=0
64804=252   64891=237   64978=253   64848=237   64935=248   65022=0
64805=211   64892=66    64979=175   64849=83    64936=253   65023=0
64806=223   64893=126   64980=184   64850=254   64937=34    65024=161
64807=251   64894=254   64981=40    64851=253   64938=254   65025=253
64808=201   64895=65    64982=14    64852=35    64939=253   65026=202
64809=14    64896=40    64983=197   64853=78    64940=119   65027=253
64810=223   64897=2     64984=205   64854=35    64941=35    65028=166
64811=237   64898=207   64985=48    64855=70    64942=1     65029=253
64812=120   64899=1     64986=253   64856=201   64943=4
64813=192   64900=35    64987=193
```

HOW TO USE THE MASTER COMMANDS

With the MASTER/SLAVE software incorporated into your computer, you have these options:

1. Save memory from the master into the slave.
2. Load memory from the slave into the master.
3. Cause the slave to quit to Basic.

The general procedure for implementing these commands requires sending a command statement to the slave which tells the slave what it is to do. Several vital pieces of information must be passed to the slave in the command statement. First, the command itself, tells the slave whether it is to send bytes, receive bytes, or quit to Basic. Second, the slave start address, specifies where in the slave's memory saving or loading is to start. Third, the number of bytes to send or receive must be given. Finally, the master start address is passed. This address indicates where in the master computer's memory saving or loading is to occur. These parameters give the slave all the information it needs: the command to execute, the starting slave address, the number of bytes to process, and the master start address.

The above parameters are passed to the slave in the form of a string of characters which has the name A$. What you do is determine the parameters you want to pass. Then you place them in A$, and RAND USR 64884 to execute the desired function. A$ must have a length of 9 characters. The first seven are used to hold the parameters needed by the machine code. Each of these seven characters has a specific role. The first character is the CHR$ number of the

command you wish to send to the slave. For the three commands implemented, the codes are 0 for SAVE, 1 for LOAD, and 2 for QUIT. For example, to save memory, you would LET A$(1)=CHR$ 0. To load memory, you LET A$(1)=CHR$ 1.

The next two characters of A$ represent the SLAVE start address where memory will be loaded or saved. These A$ characters are written as a two byte integer, with the less significant byte first. The characters are written as CHR$ numbers just like the command is. An easy way to put the proper character codes into A$(2) and A$(3) follows. Let's suppose you want the slave start address to be 64000 decimal. First, enter the command RAND 64000. This automatically converts the address into a 2 byte integer whose values are stored in the system variable, SEED. By peeking SEED, you can get the right CHR$ for A$(2) and A$(3). After you RAND 64000, you would say, LET A$(2)=CHR$ (PEEK 23670) and let A$(3)=CHR$ (PEEK 23671). These commands can be entered in the immediate mode without line numbers, or you could add program lines to handle everything automatically.

The next pair of characters in A$, the fourth and fifth, represent the number of bytes you wish to save or load. Enter these characters in exactly the same way you do the previous two characters. If, for example, you want to transfer 1000 bytes of memory, you would RAND 1000, then LET A$(4)=CHR$ (PEEK 23670), and LET A$(5)=CHR$ (PEEK 23671).

The sixth and seventh characters of A$ represent the address of the start in the master computer where saving or loading is to occur. As before, RAND the address you want, LET A$(6)= CHR$ (PEEK 23670) and LET A$(7)=CHR$ (PEEK 23671).

Two more characters must be added to A$, bringing the total length to 9. But you only need to make them blank spaces. The master operating system places a second "number of bytes" parameter in these spaces for you.

The actual job of sending data to the slave started with the command RAND USR 64884. However, in order for parameters to be passed properly, A$ must be the last variable in memory. There are many ways to arrange it thus. The simplest is probably to insert the line, LET A$=A$ just before you RAND USR 64884. As long as your final adjustment to A$ immediately precedes the RAND USR 64884 command, A$ will be the last variable in memory. What you want to avoid is a situation where some other variable is acted upon before you RAND USR. This action would cause A$ to be located in the wrong place.

SUMMARY OF PARAMETERS

To recap the parameter passing procedure, I will summarize the above and then give a few concrete programming examples. First, parameters are passed in A$. A$(1) is the function you wish to perform. If A$(1)=CHR$ 0, you SAVE memory from the master to the slave. If A$(1)=CHR$ 1, you LOAD memory from the slave to the master. A$(2 TO 3) is the start address of the slave. If you are saving to the slave, this is where the data will start. If you are loading to the master, this is the address of the first byte to go to the master computer. A$(4 TO 5) is the number of bytes to process, or the number of bytes to save or load. A$(6 TO 7) is the master start address. This is analogous to the slave address. If you are sending bytes to the slave, this address is the first byte to be sent. If you are receiving bytes from the slave, this is the address where the slave data will be sent. A$(8 TO 9) are blank spaces. Parameters must be passed in a string of the name A$, and it must be 9 characters long. Otherwise the computer will give you a REPORT 2-error.

In the case of the command which causes the slave computer to quit to Basic, the slave start address parameter is loaded into the BC register pair so that when the slave returns, this number can be used to trigger special functions. For example, you could pass the value of 1000 to the slave with the quit command. This would make the variable X equal to 1000 in the slave. You could add an additional line of basic to the slave which says something like: IF X=1000 then GOTO 5000. Then at 5000, you could add more special programming.

These save load and quit routines do not keep track of the data you transfer between computers. Often it will be very useful to be able to remember what is stored in a given location. It will be necessary to add some sort of directory or index routine in Basic to the master program to accomplish this.

PROGRAM EXAMPLES

```
TO LOAD D$ WITH SLAVE BYTES
STARTING AT 16514...
(assume LEN d$=10)

  10 LET D$="          "
  15 LET D$(1)=D$(1)
  20 RANDOMIZE PEEK 23629+256*PE
EK 23630
  30 LET A$=CHR$ 1+CHR$ 130+CHR$
 64+CHR$ 10+CHR$ 0+CHR$ (PEEK 23
670)+CHR$ (PEEK 23671)+"  "
  40 RANDOMIZE USR 64884
```

```
TO SEND CONTENTS OF D$ TO THE
SLAVE ADDRESS 32768...

   1 LET D$="THIS IS A TEST"
  10 LET A$=CHR$ 0
  20 RANDOMIZE 32768
  30 LET A$=A$+CHR$ (PEEK 23670)
+CHR$ (PEEK 23671)
  40 RANDOMIZE LEN D$
  50 LET A$=A$+CHR$ (PEEK 23670)
+CHR$ (PEEK 23671)
  60 LET D$(1)=D$(1)
  70 RANDOMIZE PEEK 23629+256*PE
EK 23630
  80 LET A$=A$+CHR$ (PEEK 23670)
+CHR$ (PEEK 23671)
  90 LET A$=A$+"  "
 100 RANDOMIZE USR 64884
```

```
TO MAKE THE SLAVE QUIT TO BASIC

  10 LET A$=CHR$ 2+CHR$ 0+CHR$ 0
+CHR$ 0+CHR$ 0+CHR$ 0+CHR$ 0+"
": STOP   REM QUIT
  20 RANDOMIZE USR 64884
```

```
TO SEND 1000 BYTES OF DATA
STARTING AT ADDRESS 16384 TO THE
SLAVE ADRESS 63000...

   1 RANDOMIZE 16384: LET M1=PEE
K 23670: LET M2=PEEK 23671
  10 RANDOMIZE 1000: LET L1=PEEK
 23670: LET L2=PEEK 23671
  20 RANDOMIZE 63000: LET A1=PEE
K 23670: LET A2=PEEK 23671
  30 LET A$=CHR$ 0+CHR$ A1+CHR$
A2+CHR$ L1+CHR$ L2+CHR$ M1+CHR$
M2+"  "
  40 RANDOMIZE USR 64884
9999 STOP
9000 LET A$=CHR$ 2+CHR$ 0+CHR$ 0
+CHR$ 0+CHR$ 0+CHR$ 0+CHR$ 0+"
": STOP : REM QUIT
```

```
TO LOAD THESE SAME 1000 BYTES
BACK INTO THE MASTER AT THEIR
ORIGINAL ADDRESSES, YOU NEED
ONLY TO CHANGE LINE 30 TO:

  30 LET A$=CHR$ 1+CHR$ A1+CHR$
A2+CHR$ L1+CHR$ L2+CHR$ M1+CHR$
M2+"  "
```

As you can see, transferring data from one computer to another requires more than just a USR call. The PEEKs and LETs in the above examples determine what, how much and which direction data is to travel. As is so often the case with computers, you can't just tell the computer to "do it" without first telling the computer what it is you want it to do. Fortunately, Basic can be used to handle these housekeeping tasks, and there are many possible ways to reach the same end. Updates will have much more to say about this master/slave system. In the meantime, you have the central workhorse routines to experiment with. Have fun, and let me know what interesting tidbits you discover.

SLAVE Disassembly

```
2000 C5        DLAY:PUSH BC      -Simple delay routine
2001 0620            LD B,20     -Set initial count to 20
2003 10FE      DLUP  DJNZ DLUP   -Then count down to zero
2005 C1              POP BC      -restore original BC value
2006 C9              RET         -then return.
2007 1600      SEND  LD D,00     -Subroutine which sends bytes
2009 0EDF            LD C,DF     -Port # is DF hex
200B 47              LD B,A      -B reg. stores # of bytes to send
200C ED79            OUT (C),A   -This number is sent to master
200E CD0020          CALL DLAY   -Then wait for master to get set.
2011 ED58      LUP1  IN E,(C)    -Master tells the slave which byte to send.
2013 2AFE20          LD HL,(BADR)-Point to start of data block to send
2016 19              ADD HL,DE   -Add the offset supplied by the master
```

```
2017 7E          LD A,(HL)      -HL now points to address of char. to send.
2018 ED79        OUT (C),A      -out it goes.
201A 7B          LD A,E         -Is the byte sent the last one?
201B B8          CP B           -Find out by comparing with B register.
201C 20F3        JR NZ LUP1     -If not, go back to LUP1 to send another byte.
201E ED78   LAST IN A,(C)       -Otherwise, wait til master is finished processing.
2020 20FC        JR NZ LAST     -This is done by just sitting until a zero comes in from master.
2022 AF     DONE XOR A          -Then send a zero out to the master, and return.
2023 D3DF        OUT (DF),A
2025 C9          RET
2026 0EDF   SLWT LD C,DF        -Slave Wait routine waits for a byte to come from the master.
2028 ED78   AGIN IN A,(C)       -It just reads the port
202A C0          RET NZ         -and returns if the byte read is not zero.
202B 18FB        JR AGIN        -Otherwise, it reads the port again.
202D CD2620 RCV1 CALL SLWT      -Subroutine to receive bytes from master. First, call the wait
2030 110000      LD DE,0000      routine. Then set DE counter to zero.
2033 47     RCV2 LD B,A         -The first byte input tells how many bytes to expect. Put it in B.
2034 ED59   LUP. OUT (C),E      -E register tells the master which byte to send. Output the value
2036 CDCA20      CALL DLA2      -then delay to allow time for the master to catch up.
2039 2AFE20      LD HL,(BADR)   -Put first address where received data is to be loaded into HL
203C 19          ADD HL,DE      -Add the offset to it.
203D ED78        IN A,(C)       -Read the byte from the master.
203F 77          LD (HL),A      -and put it into address specified by HL.
2040 78          LD A,B         -Have we received the last byte? Find out by subtracting
2041 93          SUB E          -the value in E from A
2042 13          INC DE         -increment the counter
2043 20EF        JR NZ LUP.     -If A-E <>0 then go back to receive another byte.
2045 C9          RET            -Otherwise, we're done, so return.
2046 110000 SLST LD DE,0000     -Slave Start. This is the main entry point (USR 8262).
2049 CD2620      CALL SLWT      -wait for a signal from master.
204C FE04        CP 04          -Is it a $?
204E 2805        JR Z A.OK      -If yes, it means a command is coming so proceed to read it.
2050 AF     ERR0 XOR A          -If no, its garbage so send a zero out to the master
2051 ED79        OUT (C),A
2053 18F1        JR SLST        -and go back to Slave Start.
2055 21F720 A.OK LD HL,BUFF     -First, load HL with address of command buffer (20F7 hex).
2058 22FE20      LD (BADR),HL   -Then put it in BADR variable.
205B E5          PUSH HL        -Save the register HL
205C CD3320      CALL RCV2      -and go to receive bytes, putting them in the buffer.
205F 3AF720      LD A,(BUFF)    -The first byte of the buffer is the command.
2062 210021 COMD LD HL,TABL     -HL is loaded with the address of the command table.
2065 85          ADD A,L        -Then the command offset is added
2066 6F          LD L,A
2067 CB25        SLA L          -so that HL points to the command handler location in the table.
2069 5E          LD E,(HL)      -The address in the table is loaded into DE
206A 23          INC HL
206B 56          LD D,(HL)
206C EB          EX DE,HL       -Then its put back into HL
206D D1          POP DE         -The second byte in the buffer is popped off the stack
206E 13          INC DE         -and stored in DE
206F CD2220      CALL DONE      -Send a zero out to the master.
2072 E9          JP (HL)        -and jump to the routine specified by the command.
2073 62     PRAM LD H,D         -Subroutine to interpret parameters sent by master.
2074 6B          LD L,E         -2nd and 3rd bytes in buffer  store the slave address parameter.
2075 5E          LD E,(HL)
2076 23          INC HL
2077 56          LD D,(HL)
```

```
2078 ED53FE20        LD (BADR),DE   -Put this address in BADR variable.
207C 23              INC HL         -4th and 5th bytes are the Number of bytes to send/receive parameter.
207D 4E              LD C,(HL)
207E 23              INC HL
207F 46              LD B,(HL)
2080 C9              RET            -Return with this number in BC.
2081 CD7320    BLD1  CALL PRAM      -Command handler to load a block of bytes from the master.
2084 AF              XOR A          -First, get parameters, and determine how many bytes to load.
2085 B8              CP B           -If B=0, there are less than 255
2086 280E            JR Z REM2      -so load them at REM2
2088 C5        NXBL  PUSH BC        -otherwise, load B blocks of 255 bytes each.
2089 CD2D20          CALL RCV1      -One block of 255 is loaded
208C CD2220          CALL DONE      -When complete, send a zero out to master.
208F C1              POP BC         -Get the count,
2090 23              INC HL         -Advance HL to point to the next address to load
2091 22FE20          LD (BADR),HL   -store it in BADR
2094 10F2            DJNZ NXBL      -Then loop back to load another block. This repeats til B=0.
2096 0D        REM2  DEC C          -Then the remainder is loaded. C tells how many bytes to load.
2097 28AD            JR Z SLST      -If its zero, youre done so go back to SLST
2099 CD2D20          CALL RCV1      -Otherwise, receive the bytes.
209C 1826            JR FINI        -When complete, go to FINI.
209E CD7320    BSV1  CALL PRAM      -This is the command handler to SEND bytes to the master.
20A1 AF              XOR A          -First get the parameters from the buffer just as in loading.
20A2 B8              CP B
20A3 280D            JR Z RMDR      -If less than 255 Bytes need sending, do it at RMDR.
20A5 3EFF      NXBS  LD A,FF        -Otherwise send B blocks of 255 bytes each.
20A7 C5              PUSH BC        -Save the count.
20A8 CD0720          CALL SEND      -Send the block.
20AB 23              INC HL         -Advance HL to point to next byte to send,
20AC 22FE20          LD (BADR),HL   -and put it in BADR.
20AF C1              POP BC         -Restore the count.
20B0 10F3            DJNZ NXBS      -And repeat til B=0
20B2 79        RMDR  LD A,C         -Then send the remainder. A=# of bytes to send.
20B3 FE00            CP 00          -If its zero, youre done.
20B5 288F            JR Z SLST      -So go back to SLST.
20B7 3D              DEC A          -Otherwise, send A number of bytes.
20B8 CD0720          CALL SEND
20BB 1889            JR SLST        -Then go back to SLST
```

UNCLASSIFIED

Sell that piece of gadgetry that failed the smoke test, or that extra printer, or memory pack. Non-commercial ads: $5.00 for 5 lines.

FOR SALE: Memotech Centronics Interface w/cable..$70. Memotech HRG interface $60. Both in original boxes. Call 212 535-1651 or 516 365-8737.

FOR SALE: C. ITOH model M7500 dot matrix printer. Brand new. Same speed and functions as the Prowriter but costs only $250+$7 shipping. A good light to medium duty printer. Tom Woods P.O. Box 64, Jefferson, NH 03583

COMPU/BUG Spray!! Computer error has forced the liquidation of 2000 cases of this amazing, safe, easy to use bug eliminator. Guaranteed organic, no harmful residues. FREE can for the first 500 callers. Dial: 1-800-B-U-G-O-F-F-Q Today!!!

FOR SALE: Aerco Disk System for TS/ZX complete w/2 drives, cables & DOS. $300 or trade for modem or serial LQ Printer of equal value. Jim Comperchio, Box 1347, Wolfeboro, NH 03894. (603) 569-4545.

REMEMBER to RENEW your subscription to Updates! This is your last issue for the current year. Send $12.95 to Tom Woods, P.O. Box 64, Jefferson, NH 03583

```
20BD EB        QUIT EX DE,HL      -Command handler to quit to basic.
20BE 4E             LD C,(HL)     -The master address is loaded into HL
20BF 23             INC HL        -Then into BC
20C0 46             LD B,(HL)
20C1 C32220         JP DONE       -so that the Slave USR will return the value in BC
20C4 CD2220    FINI CALL DONE     -After loading the last block of data, you come here where
20C7 C34620         JP SLST        a zero is sent to the master. Then you go back to SLST.
20CA CD0020    DLA2 CALL DLAY     -A second delay routine called from the Receive Bytes routine.
20CD CD0020         CALL DLAY     -Calls the delay routine twice for a longer pause.
20D0 C9             RET
```

MASTER Disassembly

```
6A01 C5        DLAY PUSH BC       -Simple delay routine works
6A02 0620           LD B,20       -Set B-reg. to 20
6A04 10FE           DJNZ,6A04     -Then count down to zero before
6A06 C1             POP BC         restoring original value to BC
6A07 C9             RET            and returning
6A08 F3        SEND DI            -Routine to send bytes to slave
6A09 1600           LD D,00
6A0B 0EDF           LD C,DF       -Port address=DF hex
6A0D 47             LD B,A        -B reg. stores the 1st byte sent
6A0E ED79           OUT (C),A     -Out it goes!
6A10 CD016A         CALL DLAY     -Delay to let slave catch up
6A13 ED58      LUP1 IN E,(C)      -Slave tells master which byte
6A15 2AFE68         LD HL,(BADR)   to send.
6A18 19             ADD HL,DE     -Add this offset to val. of BADR
6A19 7E             LD A,(HL)     -Put char. at (HL) in Acc.
6A1A ED79           OUT (C),A     -Send it out.
6A1C 7B             LD A,E        -Is it the last char. to send?
6A1D B8             CP B          -B=offset for last address.
6A1E 20F3           JR NZ,LUP1    -If no, go back to send another.
6A20 ED78      LAST IN A,(C)      -Otherwise, wait til slave
6A22 20FC           JR NZ,LAST     sends a zero.
6A24 D3DF      DONE OUT (DF),A    -Then send a zero to the slave
6A26 FB             EI            -reset normal interrupts
6A27 C9             RET           -and return.
6A28 0EDF      WAIT LD C,DF       -Routine to wait for char. from
6A2A ED78      AGIN IN A,(C)       the slave.
6A2C C0             RET NZ        -Return when one is received
6A2D 18FB           JR AGIN       -Else, go back to look again.
6A2F CD286A    RCV1 CALL WAIT     -RECEIVE bytes routine. First,
6A32 110000         LD DE,0000     wait for 1st char. Reset
6A35 47        RCV2 LD B,A         counter. A=# of bytes to get.
6A36 ED59           OUT (C),E     -Ask slave for a byte.
6A38 CDE96A         CALL DLA2     -wait for slave to catch up.
6A3B 2AFE68         LD HL,(BADR)  -HL+DE=addr where data is to go.
6A3E 19             ADD HL,DE
6A3F ED78           IN A,(C)      -Read the port,
6A41 77             LD (HL),A     -and put it value in memory (HL)
6A42 78             LD A,B        -Have we received the last one?
6A43 93             SUB E         -If yes, A-E equals zero.
6A44 13             INC DE        -Advance offset to next address
6A45 20EF           JR NZ,6A36    -Get more bytes if needed.
6A47 18DB           JR DONE       -If done, go to DONE.
6A49 00             NOP
6A4A 00             NOP
6A4B 00             NOP
6A4C 5E        PRAM LD E,(HL)     -Parameter routine for saving/
6A4D 23             INC HL         loading memory. (BADR) stores
6A4E 56             LD D,(HL)      addr. of 1st byte to save/load
6A4F ED53FE68       LD (BADR),DE   BC=# of bytes to process.
6A53 23             INC HL
6A54 4E             LD C,(HL)
6A55 23             INC HL
6A56 46             LD B,(HL)
6A57 C9             RET
6A58 CD4C6A    BSV1 CALL PRAM     -Routine to save a block of
6A5B AF             XOR A          memory to the slave.
6A5C B8             CP B          -If B=0 there's less than 255
6A5D 280D           JR Z,REM1      bytes to save so go to REM1.
6A5F 3EFF      NXBS LD A,FF        Otherwise send B blocks of 255
6A61 C5             PUSH BC        bytes each.
6A62 CD086A         CALL SEND
6A65 23             INC HL        -Advance HL to next address
6A66 22FE68         LD (BADR),HL  -and store it in BADR
6A69 C1             POP BC
6A6A 10F3           DJNZ,NXBS     -Loop back til all blocks sent.
```

```
6A6C 79          REM1 LD A,C          -Then send the remainder.
6A6D FE00             CP 00
6A6F C8               RET Z
6A70 3D               DEC A
6A71 1895             JR SEND
6A73 A7          MSTR AND A           -Master Entry Point (USR 64884)
6A74 2A595C           LD HL,(5C59)    -First  we check for errors.
6A77 010D00           LD BC,000D      -Start at (ELINE) and go back
6A7A ED42             SBC HL,BC        13 bytes
6A7C 7E               LD A,(HL)       -Is this the letter A? (for A$)
6A7D FE41             CP 41
6A7F 2802             JR Z,6A83       -Go on if yes.
6A81 CF          ERR0 RST 08H         -Otherwise, report Error 2
6A82 01               Error 2
6A83 23               INC HL          -Is the next byte a 0? (LEN A$)
6A84 7E               LD A,(HL)
6A85 FE09             CP 09
6A87 20F8             JR NZ,ERR0      -NO? Report Error.
6A89 23               INC HL          -Is the next byte a 0?
6A8A 7E               LD A,(HL)        (length again)
6A8B FE00             CP 00
6A8D 20F2             JR NZ,ERR0      -No? Error
6A8F 23               INC HL          -Next byte is command. Put in
6A90 7E               LD A,(HL)        Acc. and use as offset to jump
6A91 E5               PUSH HL          table starting at HL.
6A92 210069      COMD LD HL,TABL      -Calculate address in table
6A95 85               ADD A,L
6A96 6F               LD L,A
6A97 CB25             SLA L
6A99 5E               LD E,(HL)
6A9A 23               INC HL
6A9B 56               LD D,(HL)
6A9C EB               EX DE,HL
6A9D D1               POP DE
6A9E 13               INC DE          -DE=next byte of A$
6A9F E9               JP (HL)         -Jump to address of jump table.
6AA0 CDA56A      MSND CALL HEDR       -Command handler to send bytes.
6AA3 18B3             JR BSV1          First send header then data.
6AA5 21F788      HEDR LD HL,BUFF      -Header subroutine. Tell the
6AA8 22FE68           LD (BADR),HL     slave what is needed. Transfer
6AAB 77               LD (HL),A        contents of A$ to buffer area.
6AAC 23               INC HL
6AAD 010400           LD BC,0004
6AB0 EB               EX DE,HL
6AB1 EDB0             LDIR
6AB3 1B               DEC DE
6AB4 1B               DEC DE
6AB5 23               INC HL
6AB6 23               INC HL
6AB7 EB               EX DE,HL
6AB8 0E02             LD C,02
6ABA EDB0             LDIR
6ABC EB               EX DE,HL
6ABD 110400           LD DE,0004
6AC0 ED52             SBC HL,DE
6AC2 7B               LD A,E
6AC3 E5               PUSH HL
6AC4 CD086A           CALL SEND       -Then send the buffer to slave.
6AC7 E1               POP HL
6AC8 C9               RET
6AC9 CDA56A      MRCV CALL HEDR       -Command handler to receive from
6ACC 00               NOP              slave. Header goes to the slave
6ACD 00               NOP              first.
6ACE 00               NOP
6ACF CD4C6A      BLD1 CALL PRAM       -Then determine where data is
6AD2 AF               XOR A            to be placed
6AD3 B8               CP B            -Then load B blocks of data
6AD4 280E             JR Z,REM2
6AD6 C5          NXBL PUSH BC
6AD7 CD2F6A           CALL RCV1
6ADA C1               POP BC
6ADB 23               INC HL
6ADC 22FE68           LD (BADR),HL
6ADF CD016A           CALL DLAY
6AE2 10F2             DJNZ,NXBL       -Loop until done.
6AE4 0D          REM2 DEC C           -Load the last few bytes
6AE5 C42F6A           CALL NZ,RCV1    -and return
6AE8 C9               RET
6AE9 CD016A      DLA2 CALL DLAY       -Delay routine pauses twice as
6AEC CD016A           CALL DLAY        long as the other one.
6AEF C9               RET
```